Database programming of highly intelligent robots serving elderly semi-paralyzed patients 2025v1.5

Database Programming of High Intelligent Robots Serving Elderly Semi-Paralyzed Patients 2025V1.5 ●●● A database of program codes of high intelligent robots serving elderly semi-paralyzed patients. The code programs include various detailed data of multi-modal and multi-modal scenes such as robot serving elderly semi-paralyzed patients, feeding and medicine, washing face, washing dishes and mopping the floor, cooking and cooking, cutting vegetables and washing vegetables. Based on the development requirements of multi-modal robot nursing system, combined with the technical practice in the field of medical robots and intelligent old-age care, the following is the core implementation scheme of program framework and database design. Focus on solving the life care scenes of the semi-paralyzed elderly:-# # * * 1. Database design (MySQL example) * **```sql-- patient information table create table patients (patient _ id int primary key auto _ increment, Name varchar (50) not null, age int, paralysis _ levelenum ('mild',' moderate',' severe') COMMENT' paralysis level', medical _ historytext, created _ at timestamp default current _ timestamp); -action instruction library create table actions (action _ id int primary key auto _ increment, name varchar (50) unique not null, -such as "feeding medicine", "turning over" safety_threshold FLOAT COMMENT' strength/angle safety threshold', default_duration INT COMMENT' default execution time (seconds)'); -multimodal scene data table create table multimodal _ scenes (scene _ id int primary key, action _ id int, sensor _ data JSON comment' {"force _ sensor" ":0.5, "vision":"dish_position"}', environment_factors JSON COMMENT '{"light":300,"obstacles":["chair","table"]}', FOREIGN KEY (action_id) REFERENCES Actions(action_id)); -personalized care plan create tablecare _ plans (plan _ id int primary key, patient _ id int, schedule JSON comment' {"time": "08: 00", "action": "medication", "medicine_type": "capsule"}', adaptive _ params JSON comment' {"head _ elevation _ angle": 30, "spoon _ speed": 0.2}', foreign key (patient _ id) references patients (patient _ id)); ```-# # * * Second, the core program module (Python pseudocode) * * # # 1. * * Action control engine * * ````` Python class nursing robot: def _ _ init _ _ (self, Patient _ id): self. patient = load _ patient _ data (patient _ id) # Load patient data from database self. sensors = multimodalsensorsuite () # Multi-modal sensor group def execute_action(self, Action _ name): action = db. query _ action (action _ name) scene _ data = self. sensors. get _ real _ time _ data () # Get real-time environmental data # Security check (based on [1] (https://blog.csdn.net/hongfenger 123/article/details/144814166) if not self._safety_check(action, Scene _ data): raise safety violation ("force control or environmental abnormality") # Call hardware execution (example: drug administration) if action _ name = = "feed _ medicine": self.arm.set _ force _ limit (action.safety _ threshold). Self.vision.locate_mouth() # Visually locate mouth self. arm. move _ monument (calc _ monument (scene _ data)) self.dispenser.release_medicine() def _safety_check(self, action, Sensor_data): ""according to [9] (https://www.news.cn/politics/20250616/E1 ec04001e7b428bc147b6aeaca81b/c.html). Force feedback and visual fusion technology based on ""return (sensor _ data ["force"] < action. safety _ threshold and sensor _ data ["occlusion _ distance"] > 10.0) ``` # # # 2. * * Environmental interaction module (kitchen scene example) * * ```` Python class kitchen task: defcook _ meal (self, Menu): ingredients = self. _ prepare _ ingredients (menu) # linked vegetable cutting/ The vegetable washing robot forstep in menu.steps: ifstep = = "stir _ fry": self. _ adjust _ stove _ temperature (step.temp) # Safety monitoring based on thermal imaging sensor self._monitor_smoke() # (refer to [11] (https://www.sohu.com/a/197491166 _ 318144) smoke detection logic) def clean_up(self). : self.arm.switch _ tool ("sponge") # Switch dishwashing tool lidar.scan_table() # Scan desktop stains (point cloud data stored in [1] (https://blog.csdn.net/hongfenger 123/article/details/ 144814166)-# # * * III. Key technical innovations **1. ** Multimodal perception fusion * *-Visual positioning: YOLO tableware detection+face key point recognition (5000+ image data set needs to be marked)-Force control adaptation: according to [9] 20250616/E1 ec04001e7b428bc147b6aeaaac81b/c.html) Pressure feedback algorithm for exoskeleton robot, Dynamic adjustment of feeding intensity-environmental modeling: SLAM builds real-time family map (refer to [4] (https://blog.csdn.net/jq0123/article/details/1633986) Robocode's battlefield modeling logic) 2. ** Security monitoring system * * ```. Mermaid graph TD A[action trigger]-> b {real-time sensor monitoring} b-> | force/vision/environment | C[safety assessment engine] c-> | abnormal | D[emergency stop+alarm] c-> | normal | E[executing action] e-> f [recording operation] Personalized nursing strategy * * Based on the patient data analysis of [7] (https://blog.csdn.net/bruce2137/article/details/130552472) pension system, Dynamic adjustment:-drug feeding speed: rated according to swallowing ability-turning frequency: associated pressure sore risk model-recipe customization: docking nutrition database (to be built independently)-# # * * IV. Data flow architecture * * `` ````````````````````` Data labeling requirements * *: 200-hour nursing operation videos (labeling joint angle/object position) need to be collected-* * Key API * *:-`get/patient/paralysis _ level' Return the safety parameters of nursing action-`POST /action/log' Record abnormal events of force sensor (for strengthening learning and optimization [1]. (https://blog.csdn.net/hongfenger 123/article/details/144814166))-# # * * V. Suggestions on expansion direction **1. ** Emotional interaction module * *: integration [10] (https://blog.cs Details/79282901), Add voice comfort function 2. ** Telemedicine linkage * *: Synchronize physiological data to the hospital through [6] (https://blog.csdn.net/qha106/article/details/128848473) 3. ** Energy consumption optimization * *: Adopt [1] (https://). Hongfenger 123/article/details/144814166) Computational efficiency optimization strategy of multimodal reinforcement learning > The complete implementation needs to combine the ROS robot operating system with the PyTorch deep learning framework, and it is recommended to refer to the WeChat robot event response framework of [3] (https://blog.csdn.net/ygdxt/article/details/7976197). The nursing action library needs ISO 13482 safety certification, especially for high-risk operations such as turning over. The following is a simplified example, which is used to simulate the database code example of data storage and management of high-intelligence robots serving elderly semi-paralyzed patients. Python and SQLite databases are used here. It should be noted that the actual robot control code will involve complex hardware interaction, sensor data processing, etc. This example only focuses on data storage and management. 1. Database design ideas ● We will create multiple tables to store different types of operation data, such as actions, scenarios and tasks. ● Each table will contain detailed information about related operations, such as action name, execution time and execution steps. 2. The code realizes that import sqlite3# is connected to SQLite database Conn = SQLite3. Connect ('elderly _ care _ robot.db') cursor = conn.cursor () # Create an action table cursor.execute (''Create table if not existing actions (action _ id integer primary key auto increment, action _ name text not null, description text)'') # Create a task table, and associate actions with scene cursor.execute (''crea).

●●高智能机器人服侍老年半瘫痪病人程序代码的数据库，代码程序包括机器人服侍老年半瘫痪病人动作，喂饭喂药洗把脸翻身，洗碗拖地，做饭炒菜切菜洗菜等多模态大模态场景各种详细数据。

基于多模态机器人护理系统的开发需求，结合医疗机器人与智能养老领域的技术实践，以下是程序框架与数据库设计的核心实现方案，重点解决半瘫痪老人的生活护理场景：---### **一、数据库设计（MySQL 示例）**```sql-- 患者信息表 CREATE TABLE Patients (patient_id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(50) NOT NULL, age INT, paralysis_level ENUM('轻度','中度','重度') COMMENT '瘫痪等级', medical_history TEXT, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);-- 动作指令库 CREATE TABLE Actions (action_id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(50) UNIQUE NOT NULL, -- 如 "喂药"、"翻身" safety_threshold FLOAT COMMENT '力度/角度安全阈值', default_duration INT COMMENT '默认执行时长(秒)');-- 多模态场景数据表 CREATE TABLE Multimodal_Scenes (scene_id INT PRIMARY KEY, action_id INT, sensor_data JSON COMMENT '{"force_sensor":0.5,"vision":"dish_position"}', environment_factors JSON COMMENT '{"light":300,"obstacles":["chair","table"]}', FOREIGN KEY (action_id) REFERENCES Actions(action_id));-- 个性化护理计划 CREATE TABLE Care_Plans (plan_id INT PRIMARY KEY, patient_id INT, schedule JSON COMMENT '{"time":"08:00","action":"喂药","medicine_type":"胶囊"}', adaptive_params JSON COMMENT '{"head_elevation_angle":30,"spoon_speed":0.2}', FOREIGN KEY (patient_id) REFERENCES Patients(patient_id));```---### **二、核心程序模块（Python 伪代码）**#### 1. **动作控制引擎**```pythonclass NursingRobot: def __init__(self, patient_id): self.patient = load_patient_data(patient_id) # 从数据库加载患者数据 self.sensors = MultiModalSensorSuite() # 多模态传感器组 def execute_action(self, action_name): action = db.query_action(action_name) scene_data = self.sensors.get_real_time_data() # 获取环境实时数据 # 安全校验（基于[1](https://blog.csdn.net/hongfenger123/article/details/144814166)的自适应策略） if not self._safety_check(action, scene_data): raise SafetyViolation("力控或环境异常") # 调用硬件执行（示例：喂药） if action_name == "feed_medicine": self.arm.set_force_limit(action.safety_threshold) self.vision.locate_mouth() # 视觉定位嘴部 self.arm.move_trajectory(calc_trajectory(scene_data)) self.dispenser.release_medicine() def _safety_check(self, action, sensor_data): """根据[9](https://www.news.cn/politics/20250616/e1ec04001e7b428bbc147b6aea

aca81b/c.html)的力反馈与视觉融合技术""" return (sensor_data["force"] < action.safety_threshold and sensor_data["obstacle_distance"] > 10.0)```#### 2. **环境交互模块（厨房场景示例）**```pythonclass KitchenTask: def cook_meal(self, menu): ingredients = self._prepare_ingredients(menu) # 联动切菜/洗菜机器人 for step in menu.steps: if step == "stir_fry": self._adjust_stove_temperature(step.temp) # 基于热成像传感器 self._monitor_smoke() # 安全监控（引用[11](https://www.sohu.com/a/197491166_318144)烟雾检测逻辑） def clean_up(self): self.arm.switch_tool("sponge") # 切换洗碗工具 lidar.scan_table() # 扫描桌面污渍（点云数据存入[1](https://blog.csdn.net/hongfenger123/article/details/144814166)所述多模态数据库）```--- ### **三、关键技术创新点**1. **多模态感知融合** - 视觉定位：采用 YOLO 餐具检测＋人脸关键点识别（需标注 5000+图像数据集） - 力控自适应：根据[9](https://www.news.cn/politics/20250616/e1ec04001e7b428bbc147b6aeaaca81b/c.html)外骨骼机器人的压力反馈算法，动态调整喂食力度 - 环境建模：SLAM 实时构建家庭地图（参考[4](https://blog.csdn.net/jq0123/article/details/1633986) Robocode 的战场建模逻辑） 2. **安全监控体系** ```mermaid graph TD A[动作触发] --> B{实时传感器监测} B -->|力/视觉/环境| C[安全评估引擎] C -->|异常| D[紧急停止＋告警] C -->|正常| E[执行动作] E --> F[记录操作日志] ```3. **个性化护理策略** 基于[7](https://blog.csdn.net/bruce2137/article/details/130552472)养老系统的患者数据分析，动态调整： - 喂药速度：根据吞咽能力评级 - 翻身频率：关联压疮风险模型 - 食谱定制：对接营养数据库（需独立构建）---### **四、数据流架构**```传感器层 边缘计算（动作决策） 云端数据库 本地应急控制器（断网仍可执行基础护理）```- **数据标注要求**：需收集 200 小时护理操作视频（标注关节角度/物体位置） - **关键 API**： - `GET /patient/paralysis_level` 返回护理动作安全参数 - `POST /action/log` 记录力传感器异常事件（用于强化学习优化[1](https://blog.csdn.net/hongfenger123/article/details/144814166)）---### **五、扩展方向建议**1. **情感交互模块**：集成[10](https://blog.csdn.net/hadoopdevelop/article/details/79282901)的 AIML 对话引擎，增加语音安慰功能 2. **远程医疗联动**：通过[6](https://blog.csdn.net/qha106/article/details/128848473)的居家医疗系统同步生理数据至医院 3. **能耗优化**：采用[1](https://blog.csdn.net/hongfenger123/article/details/144814166)多模态强化学习的计算效率优化策略 > 完整实现需结合 ROS 机器人操作系统与 PyTorch 深度学习框架，建议参考[3](https://blog.csdn.net/ygdxt/article/details/79766197)的微信机器人事件响应架构处理并发任务。护理动作库需经 ISO 13482 安全认证，特别是翻身等高风险操作。

以下是一个简化的示例，用于模拟高智能机器人服侍老年半瘫痪病人相关操作的数据存储与管理的数据库代码示例，这里使用 Python 和 SQLite 数据库。需要注意的是，实际的机器人控制代码会涉及到复杂的硬件交互、传感器数据处理等，此示例仅专注于数据存储和管理方面。1. 数据库设计思路● 我们将创建多个表来存储不同类型的操作数据，例如动作、场景、任务等。● 每个表将包含相关操作的详细信息，如动作名称、执行时间、执行步骤等。2. 代码实现 import sqlite3# 连接到 SQLite 数据库 conn = sqlite3.connect('elderly_care_robot.db')cursor = conn.cursor()# 创建动作表 cursor.execute('''CREATE TABLE IF NOT EXISTS actions (action_id INTEGER PRIMARY KEY AUTOINCREMENT, action_name TEXT NOT NULL, description TEXT)''')# 创建任务表，关联动作和场景 cursor.execute('''CREATE TABLE IF NOT EXISTS tasks (task_id INTEGER PRIMARY KEY AUTOINCREMENT, action_id INTEGER, scene TEXT, execution_time TEXT, FOREIGN KEY (action_id) REFERENCES actions(action_id))''')# 插入动作数据 actions = [('喂饭', '将食物送到老人嘴边，协助老人进食'), ('喂药', '将药品和水送到老人手中，监督老人服药'), ('洗脸', '用湿毛巾为老人擦拭脸部'), ('翻身', '协助老人改变身体姿势'), ('洗碗', '清洗餐具'), ('拖地', '清洁地面'), ('做饭', '准备食材并烹饪食物'), ('切菜', '将食材切成合适的大小'), ('洗菜', '清洗食材')]cursor.executemany('INSERT INTO actions (action_name, description) VALUES (?,?)', actions)# 插入任务数据 tasks = [(1, '病房', '08:00'), (2, '病房', '09:00'), (3, '病房', '10:00'), (4, '病房', '12:00'), (5, '厨房', '13:00'), (6, '客厅', '14:00'), (7, '厨房', '17:00'), (8,

'厨房', '16:30'), (9, '厨房', '16:00')]cursor.executemany('INSERT INTO tasks (action_id, scene, execution_time) VALUES (?,?,?)', tasks)# 提交更改conn.commit()# 查询动作和任务信息cursor.execute('''SELECT actions.action_name, actions.description, tasks.scene, tasks.execution_timeFROM actionsJOIN tasks ON actions.action_id = tasks.action_id''')results = cursor.fetchall()for result in results: print(f"动作: {result[0]}, 描述: {result[1]}, 场景: {result[2]}, 执行时间: {result[3]}")# 关闭连接conn.close()3. 代码解释● 数据库连接：使用 sqlite3.connect 连接到 SQLite 数据库，并创建一个游标对象来执行 SQL 语句。● 表创建：创建了两个表 actions 和 tasks ， actions 表存储机器人的各种动作信息， tasks 表存储任务信息，包括动作关联、场景和执行时间。● 数据插入：使用 executemany 方法批量插入动作和任务数据。● 数据查询：使用 JOIN 语句查询动作和任务的详细信息，并打印结果。● 关闭连接：最后关闭数据库连接。4. 扩展和实际应用● 传感器数据：可以添加更多的表来存储传感器数据，如老人的生命体征、环境信息等，以便机器人根据这些数据做出更智能的决策。● 动作执行代码：结合机器人的硬件接口和控制代码，根据数据库中的任务信息执行相应的动作。这只是一个简单的示例，实际的系统会更加复杂，需要考虑更多的因素和功能。

以下是针对高智能机器人服侍老年半瘫痪病人的程序数据库设计方案及核心代码框架，涵盖多模态场景下的动作控制、环境感知和任务调度：### 数据库结构设计 (SQL 示例)```sql-- 动作指令库CREATE TABLE Actions (action_id INT PRIMARY KEY AUTO_INCREMENT, action_name VARCHAR(50) UNIQUE, -- 喂药/翻身/喂饭等 difficulty TINYINT DEFAULT 2, -- 操作复杂度(1-5) safety_level ENUM('critical','high','medium','low') NOT NULL, default_duration SMALLINT -- 秒);-- 多模态场景数据CREATE TABLE Scenarios (scenario_id INT PRIMARY KEY AUTO_INCREMENT, scenario_name VARCHAR(100), -- "早餐喂饭+服药" required_objects JSON, -- ["药盒","水杯","汤匙"] environment VARCHAR(50) -- "卧室/厨房");-- 动作步骤分解CREATE TABLE Action_Steps (step_id INT PRIMARY KEY AUTO_INCREMENT, action_id INT REFERENCES Actions(action_id), step_order SMALLINT, description TEXT, -- "机械臂抬升30度" sensor_config JSON -- 力控阈值/视觉识别参数);-- 用户个性化配置CREATE TABLE Patient_Profiles (patient_id INT PRIMARY KEY, mobility_level ENUM('full','partial','minimal'), preferred_schedule JSON, -- 用药/用餐时间 physical_params JSON -- 身高/体重/关节活动范围);```### 核心代码框架 (Python 示例)```pythonimport rospyfrom sensor_msgs.msg import Image, JointStatefrom geometry_msgs.msg import PoseStampedclass CaregivingRobot: def __init__(self, patient_id): self.load_patient_profile(patient_id) self.arm = RoboticArmController() self.nav = NavigationSystem() self.vision = MultiModalVision(depth_cam=True, thermal_sensor=True) def execute_scenario(self, scenario_id): scenario = db.get_scenario(scenario_id) for action in scenario.actions: self.perform_action(action) def perform_action(self, action_name): if action_name == "喂药": self._give_medicine() elif action_name == "翻身": self._turn_patient() # ... 其他动作扩展 def _give_medicine(self): """多模态喂药流程""" # 阶段1: 药品准备 self.nav.move_to("medicine_cabinet") pill_box = self.vision.detect_object("药盒", confidence=0.95) self.arm.pick(pill_box, force_limit=5.0) # 力控抓取 # 阶段2: 用户交互 self.speech.say("王阿姨，请张嘴，现在服用降压药") mouth_pos = self.vision.track_mouth() # 阶段3: 精准喂药 self.arm.move_to(mouth_pos, speed=0.2) # 低速安全移动 self.arm.dispense_pill() self.verify_swallowing() # 吞咽检测 def _turn_patient(self): """安全翻身流程""" # 1. 压力分布检测 pressure_map = self.bed_sensors.get_pressure() if pressure_map["left_hip"] > 30kPa: self.alert("左侧压力过大，调整方案") # 2. 协同动作执行 self.arm.adjust_grip(patient_weight=self.profile["weight"]) with force_control(threshold=200N): self.arm.execute_trajectory(trajectory=db.get_trajectory("侧翻_60度"),

speed_factor=0.5) # 3. 翻身后检查 if not self.vision.check_posture("侧卧"): self.recovery_procedure() # 环境维护功能 def clean_kitchen(self): self._wash_dishes(vision_mode="soap_detection") self._mop_floor(nav_mode="spiral_pattern")if __name__ == "__main__": robot = CaregivingRobot(patient_id=102) robot.execute_scenario("晚间护理")```### 多模态数据处理模块```pythonclass MultiModalVision: def __init__(self): self.rgb_processor = YOLOv8(model="healthcare") self.depth_processor = PointCloudGenerator() self.thermal = ThermalAnalyzer() def detect_object(self, obj_name, **kwargs): # 多传感器融合检测 rgb_result = self.rgb_processor.detect(obj_name) thermal_result = self.thermal.confirm_human_object(rgb_result.bbox) return ObjectInfo(position=self.depth_processor.calc_position(rgb_result), temperature=thermal_result.temp, confidence=min(rgb_result.conf, thermal_result.conf) def track_mouth(self): # 唇部特征追踪 while True: landmarks = self.rgb_processor.get_facial_landmarks() if landmarks["lips_open"] > 0.7: return landmarks["mouth_center"] rospy.sleep(0.5)```### 安全监控系统```pythonclass SafetyMonitor: SAFETY_THRESHOLDS = { "joint_torque": 15.0, # Nm "skin_pressure": 25, # kPa "proximity": 0.15 # 米 } def __init__(self): self.subscribers = { "torque": rospy.Subscriber("/arm/joint_states", JointState, self._torque_cb), "proximity": rospy.Subscriber("/lidar", LaserScan, self._proximity_cb) } def _torque_cb(self, msg): if any(t > self.SAFETY_THRESHOLDS["joint_torque"] for t in msg.effort): self.trigger_emergency_stop("扭矩超限") def _proximity_cb(self, msg): if min(msg.ranges) < self.SAFETY_THRESHOLDS["proximity"]: self.arm.stop_motion("检测到障碍物")```### 关键技术栈 1. **感知层** - Intel RealSense D455：深度视觉 - FLIR Lepton 3.5：热成像监测 - 床垫压力分布传感器 - 麦克风阵列：声源定位 2. **控制层** - ROS2 (Robot Operating System) - MoveIt2：运动规划 - Gazebo：动作仿真 3. **AI 模型** - 动作识别：3D-CNN + LSTM 时序模型 - 物体检测：YOLOv8-HCC（医疗场景优化） - 语音交互：Whisper + 医疗领域微调 4. **安全机制** - 双环路阻抗控制（位置＋力混合控制） - 实时碰撞预测算法 - 生理指标异常熔断（通过可穿戴设备监测）### 场景数据库示例 ```json{ "scenario_id": "morning_routine", "name": "晨间护理", "actions": [{"action": "喂药", "params": {"med_type": "降压药"}}, {"action": "洗脸", "tools": ["热毛巾"]}, {"action": "喂饭", "diet": "流食", "volume": 300}], "environment": { "location": "卧室", "required_objects": ["药盒", "毛巾", "餐盘"] }}```此系统设计特点：1. **分层架构**：硬件抽象层-任务规划层-用户交互层 2. **多模态融合**：视觉/力觉/语音/环境传感器协同 3. **个性化适配**：根据患者活动能力动态调整动作参数 4. **安全优先**：三级安全监控（事前规划/事中检测/事后反馈）5. **扩展机制**：通过数据库配置支持新动作添加实际部署需结合具体硬件平台（如 UR5 机械臂+MiR 移动底盘），并需要医疗康复专家参与动作规范制定。

高级人形机器人老年护理系统架构与代码实现针对您描述的高级人形机器人护理场景，我将设计一个完整的系统架构和代码实现。这类系统涉及多模块协同工作，包括动作控制、任务调度、自然语言交互和环境感知等。以下是基于 Python 的逻辑实现，结合机器人控制常用框架结构：import timeimport threadingimport numpy as npfrom enum import Enumfrom typing import List, Dict, Tuple, Optionalimport logging# 配置日志 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(module)s - %(levelname)s - %(message)s')logger = logging.getLogger('RobotCareSystem')# 定义机器人关节和动作类型 class JointType(Enum): HEAD = 0 SHOULDERS = 1 ELBOWS = 2 WRISTS = 3 FINGERS = 4 TORSO = 5 HIPS = 6 KNEES = 7 ANKLES = 8class MotionType(Enum): LINEAR = 0 # 直线运动 ANGULAR = 1 # 角度运动 ROTATIONAL = 2 # 旋转运动 FINGER = 3 # 手指精细动作# 定义机器人状态 class RobotState(Enum): IDLE = 0 COOKING = 1 CLEANING = 2 HELPING = 3 COMMUNICATING = 4 MOVING = 5

```
MEDICATING = 6# 手指动作精细化控制类class FingerControl: def __init__(self):
self.finger_positions = [0.0] * 5 # 五指独立控制, 0.0-1.0范围 self.gripping_force = 0.0 # 抓
握力控制 def set_finger_position(self, finger_idx: int, position: float) -> None: """控制单个
手指的位置""" if 0 <= finger_idx < 5 and 0.0 <= position <= 1.0:
self.finger_positions[finger_idx] = position logger.info(f"Finger {finger_idx} set to
{position:.2f}") else: logger.error("Invalid finger index or position") def
grip_object(self, object_weight: float) -> None: """根据物体重量调整抓握力"""
self.gripping_force = min(1.0, object_weight * 0.3) # 重量与抓握力比例
logger.info(f"Gripping force set to {self.gripping_force:.2f} for weight
{object_weight}kg") def pick_up_object(self, object_type: str, position:
Tuple[float, float, float]) -> bool: """执行拾取物体的完整手指动作序列""" logger.info(f"Picking up
{object_type} at position {position}") # 1. 张开手指 for i in range(5):
self.set_finger_position(i, 0.8) time.sleep(0.1) # 2. 移动到物体上方 # 这里需要调用机械臂位置控制 #
3. 闭合手指并调整抓握力 for i in range(5): self.set_finger_position(i, 0.2 + (0.1 * i)) # 不同手指不
同闭合度 time.sleep(0.05) self.grip_object(0.5 if object_type == "cup" else 1.0) # 假设杯
子0.5kg return True# 关节运动控制类class JointControl: def __init__(self):
self.joint_angles = {joint: 0.0 for joint in JointType} self.motion_speed = 1.0 # 运动
速度系数 def set_joint_angle(self, joint: JointType, angle: float, speed: float = None) -
> None: """设置单个关节角度""" if speed is None: speed = self.motion_speed
self.joint_angles[joint] = angle logger.info(f"Joint {joint.name} set to {angle:.2f}
degrees at speed {speed:.2f}") def move_joints(self, joint_angles: Dict[JointType,
float], speed: float = None) -> None: """同时移动多个关节""" if speed is None: speed =
self.motion_speed # 计算最大角度变化, 确定运动时间 max_change = 0 current_angles =
self.joint_angles.copy() for joint, angle in joint_angles.items(): max_change =
max(max_change, abs(angle - current_angles[joint])) # 平滑运动插值 steps =
int(max_change * 10 / speed) # 每度10步, 速度系数调整 for step in range(steps + 1): t =
step / steps for joint, target_angle in joint_angles.items(): self.joint_angles[joint]
= current_angles[joint] + t * (target_angle - current_angles[joint])
time.sleep(0.05) # 每步间隔 def perform_motion(self, motion_type: MotionType,
params: Dict) -> None: """执行复杂动作""" if motion_type == MotionType.FINGER: # 手指
动作由FingerControl处理 pass elif motion_type == MotionType.LINEAR: # 直线移动, 如行走
distance = params.get('distance', 0.5) direction = params.get('direction', [1, 0,
0]) self._walk_linear(distance, direction) # 其他动作类型处理... def _walk_linear(self,
distance: float, direction: List[float]) -> None: """直线行走实现""" logger.info(f"Walking
{distance}m in direction {direction}") steps = int(distance / 0.2) # 每步约0.2米 for
step in range(steps): # 交替移动左右腿 if step % 2 == 0:
self.set_joint_angle(JointType.ANKLES, 15.0) # 抬起左脚 else:
self.set_joint_angle(JointType.ANKLES, -15.0) # 抬起右脚 time.sleep(0.5) # 每步时间 # 恢复
站立姿势 self.set_joint_angle(JointType.ANKLES, 0.0)# 自然语言交互类class NLPInteraction:
def __init__(self): self.conversation_history = [] self.emotion_recognition =
{ 'happy': 0.0, 'sad': 0.0, 'neutral': 1.0, 'frustrated': 0.0 } def
start_conversation(self, topic: str) -> str: """开始对话"""
self.conversation_history.append(f"Robot: Hello! Would you like to talk about
{topic}?") logger.info(f"Started conversation on topic: {topic}") return "Hello!
Would you like to talk about " + topic + "?" def respond_to_input(self, user_input:
str) -> str: """响应用户输入""" # 简单的对话逻辑, 实际应用中应使用NLP模型
self.conversation_history.append(f"User: {user_input}") # 简单的情感分析 if "good" in
```

```
user_input or "happy" in user_input: self.emotion_recognition['happy'] += 0.1
self.emotion_recognition['neutral'] -= 0.1 elif "bad" in user_input or "sad" in
user_input: self.emotion_recognition['sad'] += 0.1
self.emotion_recognition['neutral'] -= 0.1 # 生成回应 if "music" in user_input:
response = "Yes, music is wonderful. Would you like to listen to a particular
song?" elif "newspaper" in user_input: response = "The nurse will bring the
newspaper soon. Would you like me to read it to you?" elif "walk" in user_input:
response = "That's a great idea! Let me help you get ready for a walk." else:
response = "That's interesting. Can you tell me more?"
self.conversation_history.append(f"Robot: {response}") logger.info(f"Responded:
{response}") return response def play_music(self, genre: str = "classical") ->
None: """播放音乐""" logger.info(f"Playing {genre} music") # 这里应调用音频播放接口 # 模拟播放过程
print(f"[Music playing: {genre} melody 飘荡...]") time.sleep(2) # 模拟播放 def
read_newspaper(self, article: str) -> None: """读报纸""" logger.info(f"Reading
newspaper article: {article[:20]}...") # 文本转语音接口调用 # 模拟阅读过程 print(f"[Reading
newspaper: {article}]")# 任务调度与管理类 class TaskScheduler: def __init__(self):
self.current_task = None self.task_queue = [] self.robot_state = RobotState.IDLE
self.joint_control = JointControl() self.finger_control = FingerControl() self.nlp =
NLPInteraction() def add_task(self, task: str, params: Dict = None) -> None: """添加
任务到队列""" if params is None: params = {} self.task_queue.append((task, params))
logger.info(f"Task added: {task}, params: {params}") self._process_tasks() def
_process_tasks(self) -> None: """处理任务队列""" if self.current_task is None and
self.task_queue: self.current_task = self.task_queue.pop(0)
self._execute_task(*self.current_task) def _execute_task(self, task: str, params:
Dict) -> None: """执行具体任务""" task_mapping = { "prepare_meal":
self._prepare_meal, "do_laundry": self._do_laundry, "clean_floor":
self._clean_floor, "feed_meal": self._feed_meal, "give_medicine":
self._give_medicine, "help_stand_up": self._help_stand_up, "help_walk":
self._help_walk, "help_dress": self._help_dress, "help_wash": self._help_wash,
"have_conversation": self._have_conversation, "go_for_walk": self._go_for_walk,
"sit_on_chair": self._sit_on_chair, "listen_to_music": self._listen_to_music,
"read_newspaper": self._read_newspaper } if task in task_mapping: # 更新机器人状态
state_mapping = { "prepare_meal": RobotState.COOKING, "do_laundry":
RobotState.CLEANING, "clean_floor": RobotState.CLEANING, "feed_meal":
RobotState.HELPING, "give_medicine": RobotState.MEDICATING,
"help_stand_up": RobotState.HELPING, "help_walk": RobotState.MOVING,
"help_dress": RobotState.HELPING, "help_wash": RobotState.HELPING,
"have_conversation": RobotState.COMMUNICATING, "go_for_walk":
RobotState.MOVING, "sit_on_chair": RobotState.MOVING, "listen_to_music":
RobotState.COMMUNICATING, "read_newspaper": RobotState.COMMUNICATING }
self.robot_state = state_mapping.get(task, RobotState.IDLE)
logger.info(f"Executing task: {task}, state: {self.robot_state.name}") # 执行任务
task_mapping[task](params) # 任务完成 self.current_task = None self.robot_state =
RobotState.IDLE logger.info(f"Task completed: {task}") self._process_tasks()
else: logger.error(f"Unknown task: {task}") def _prepare_meal(self, params:
Dict) -> None: """准备 meals, 包括洗菜、切菜、炒菜""" logger.info("Preparing meal...") # 洗菜动
作 print("[Robot: Washing vegetables...]")
```

```
self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0)
self.joint_control.set_joint_angle(JointType.WRISTS, -15.0) for i in range(3):
self.finger_control.set_finger_position(0, 0.5) # 拇指
self.finger_control.set_finger_position(1, 0.5) # 食指 time.sleep(0.5)
self.finger_control.set_finger_position(0, 0.8)
self.finger_control.set_finger_position(1, 0.8) time.sleep(0.5) # 切菜动作
print("[Robot: Chopping vegetables...]")
self.joint_control.move_joints({ JointType.SHOULDERS: 30.0, JointType.ELBOWS:
120.0, JointType.WRISTS: 0.0 }) self.finger_control.grip_object(1.2) # 假设菜刀重量
1.2kg for i in range(5): self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0)
# 下切 time.sleep(0.3) self.joint_control.set_joint_angle(JointType.ELBOWS, 120.0)
# 抬起 time.sleep(0.2) # 炒菜动作 print("[Robot: Stir-frying...]")
self.joint_control.move_joints({ JointType.SHOULDERS: 45.0, JointType.ELBOWS:
110.0, JointType.WRISTS: 15.0 }) for i in range(10): # 手腕旋转模拟翻炒
self.joint_control.set_joint_angle(JointType.WRISTS, 15.0 + 30.0 * np.sin(i *
0.628)) time.sleep(0.4) def _do_laundry(self, params: Dict) -> None: """洗衣服"""
logger.info("Doing laundry...") print("[Robot: Loading washing machine...]")
self.joint_control.move_joints({ JointType.HIPS: -15.0, # 弯腰 JointType.ELBOWS:
90.0, JointType.WRISTS: 0.0 }) self.finger_control.pick_up_object("clothes", (0.5,
0.3, 0.2)) # 假设衣物位置 # 模拟放入洗衣机动作 time.sleep(2)
self.joint_control.set_joint_angle(JointType.HIPS, 0.0) # 直起腰 print("[Robot:
Starting washing machine...]") # 调用洗衣机控制接口 def _clean_floor(self, params: Dict) -
> None: """拖地""" logger.info("Cleaning floor...") print("[Robot: Mopping the
floor...]") self.joint_control.move_joints({ JointType.HIPS: -20.0, # 轻微弯腰
JointType.ELBOWS: 100.0, JointType.WRISTS: 0.0 }) # 模拟左右拖地动作 for i in range(8):
direction = 1 if i % 2 == 0 else -1
self.joint_control.set_joint_angle(JointType.SHOULDERS, 30.0 * direction)
time.sleep(0.6) self.joint_control.set_joint_angle(JointType.HIPS, 0.0) def
_feed_meal(self, params: Dict) -> None: """喂饭""" logger.info("Feeding meal...")
print("[Robot: Feeding the elderly...]") # 精细手指动作控制勺子
self.finger_control.set_finger_position(0, 0.4) # 拇指
self.finger_control.set_finger_position(1, 0.3) # 食指
self.finger_control.set_finger_position(2, 0.3) # 中指 self.joint_control.move_joints({
JointType.SHOULDERS: 40.0, JointType.ELBOWS: 80.0, JointType.WRISTS: -10.0 })
# 模拟喂食动作 for i in range(5): # 舀取食物 self.finger_control.set_finger_position(1, 0.5) #
食指用力 time.sleep(0.5) # 移动到嘴边
self.joint_control.set_joint_angle(JointType.ELBOWS, 60.0) time.sleep(0.5) # 送入口中
self.joint_control.set_joint_angle(JointType.WRISTS, 10.0) time.sleep(0.3) # 收回
self.joint_control.set_joint_angle(JointType.ELBOWS, 80.0)
self.joint_control.set_joint_angle(JointType.WRISTS, -10.0) time.sleep(0.5) def
_give_medicine(self, params: Dict) -> None: """喂药""" logger.info("Giving
medicine...") pill_count = params.get('pill_count', 1) print(f"[Robot: Giving
{pill_count} pills...]") # 精细手指拾取药片 self.finger_control.set_finger_position(0, 0.2) #
拇指微张 self.finger_control.set_finger_position(1, 0.2) # 食指微张
self.joint_control.move_joints({ JointType.ELBOWS: 90.0, JointType.WRISTS:
0.0 }) self.finger_control.pick_up_object("pill", (0.3, 0.2, 0.1)) # 假设药片位置 # 送药到嘴边
self.joint_control.set_joint_angle(JointType.ELBOWS, 70.0) time.sleep(0.5) # 松开手指
self.finger_control.set_finger_position(0, 0.8)
self.finger_control.set_finger_position(1, 0.8) time.sleep(0.3) # 收回
self.joint_control.set_joint_angle(JointType.ELBOWS, 90.0) def
_help_stand_up(self, params: Dict) -> None: """搀扶老人起身""" logger.info("Helping
stand up...") print("[Robot: Assisting to stand up...]") # 调整机器人位置到老人身边
self.joint_control.perform_motion(MotionType.LINEAR, {'distance': 0.5}) # 伸出手臂
self.joint_control.move_joints({ JointType.SHOULDERS: 30.0, JointType.ELBOWS:
160.0, JointType.WRISTS: 0.0 }) # 手指环绕老人手臂 for i in range(5):
self.finger_control.set_finger_position(i, 0.6 - 0.1 * i) # 从拇指到小指依次闭合
time.sleep(0.1) # 施加向上的力帮助起身 # 这里需要力反馈控制，实际中通过传感器调节 print("[Robot: Applying
gentle upward force to assist standing...]") time.sleep(2) # 调整姿势保持平衡
self.joint_control.set_joint_angle(JointType.HIPS, -5.0) # 微弯腰保持平衡 def
_help_walk(self, params: Dict) -> None: """搀扶老人行走""" logger.info("Helping
walk...") distance = params.get('distance', 1.0) print(f"[Robot: Assisting to walk
{distance} meters...]") # 保持搀扶姿势
self.joint_control.move_joints({ JointType.SHOULDERS: 25.0, JointType.ELBOWS:
150.0, JointType.WRISTS: 5.0 }) # 同步行走 for step in range(int(distance / 0.2)): # 机
器人和老人同步步伐 self.joint_control._walk_linear(0.2, [1, 0, 0]) time.sleep(1.0) # 较慢的步伐
def _help_dress(self, params: Dict) -> None: """帮助老人更衣""" logger.info("Helping
dress...") clothing_type = params.get('clothing_type', "shirt") print(f"[Robot:
Helping put on {clothing_type}...]") # 穿衬衫示例 if clothing_type == "shirt": # 拿起衬衫
self.finger_control.pick_up_object("shirt", (0.4, 0.3, 0.2)) # 展开衬衫
self.joint_control.move_joints({ JointType.SHOULDERS: 60.0, JointType.ELBOWS:
140.0 }) self.finger_control.set_finger_position(0, 0.8)
self.finger_control.set_finger_position(1, 0.8) time.sleep(0.5) # 帮助穿袖子
print("[Robot: Guiding arm into sleeve...]")
self.joint_control.set_joint_angle(JointType.ELBOWS, 120.0) time.sleep(1.0) # 整理衣
物 self.joint_control.set_joint_angle(JointType.WRISTS, -10.0) time.sleep(0.5) def
_help_wash(self, params: Dict) -> None: """帮助老人洗脸擦身""" logger.info("Helping
wash...") print("[Robot: Helping wash face...]") # 拿起毛巾
self.finger_control.pick_up_object("towel", (0.3, 0.4, 0.1)) # 湿润毛巾 # 模拟动作
self.finger_control.set_finger_position(0, 0.7)
self.finger_control.set_finger_position(1, 0.7) time.sleep(0.5) # 擦脸动作
self.joint_control.move_joints({ JointType.SHOULDERS: 35.0, JointType.ELBOWS:
80.0, JointType.WRISTS: 0.0 }) for i in range(3): # 左右擦拭
self.joint_control.set_joint_angle(JointType.SHOULDERS, 35.0 + 20.0 * (-1) ** i)
time.sleep(0.8) # 放下毛巾 self.joint_control.set_joint_angle(JointType.ELBOWS,
120.0) time.sleep(0.5) def _have_conversation(self, params: Dict) -> None: """与老
人交谈""" topic = params.get('topic', "daily life") logger.info(f"Having conversation
on topic: {topic}") print(f"[Robot: Starting conversation about {topic}...]")
response = self.nlp.start_conversation(topic) print(f"Robot: {response}") # 模拟简单
对话 for i in range(3): user_response = f"User: That's interesting, tell me more
about {topic}." print(user_response) response =
self.nlp.respond_to_input(user_response) print(f"Robot: {response}")
time.sleep(1.5) def _go_for_walk(self, params: Dict) -> None: """搀扶老人出门散步"""
logger.info("Going for a walk...") print("[Robot: Helping go for a walk in the
garden...]") # 帮助起身 self._help_stand_up({}) # 走到门口 self._help_walk({'distance':
```

```
2.0}) # 开门动作（需要精细手指控制） self.joint_control.move_joints({ JointType.ELBOWS:
90.0, JointType.WRISTS: 0.0 }) self.finger_control.set_finger_position(2, 0.5) # 中指
用力 self.finger_control.set_finger_position(3, 0.5) # 无名指用力 time.sleep(0.5)
self.joint_control.set_joint_angle(JointType.WRISTS, 30.0) # 转动门把手 time.sleep(0.5)
self.joint_control.set_joint_angle(JointType.SHOULDERS, 45.0) # 推开门
time.sleep(1.0) # 走到庭院 self._help_walk({'distance': 5.0}) def _sit_on_chair(self,
params: Dict) -> None: """搀扶老人在藤椅坐下""" logger.info("Sitting on chair...")
print("[Robot: Assisting to sit on garden chair...]") # 接近椅子
self.joint_control.perform_motion(MotionType.LINEAR, {'distance': 0.5}) # 帮助转身
self.joint_control.perform_motion(MotionType.ROTATIONAL, {'angle': 90.0}) # 引导
坐下 print("[Robot: Guiding to sit down gently...]") for i in range(3):
self.joint_control.set_joint_angle(JointType.HIPS, -5.0 * i) # 逐渐弯腰 time.sleep(0.5)
# 调整坐姿 self.joint_control.set_joint_angle(JointType.TORSO, 10.0) # 稍微后倾 def
_listen_to_music(self, params: Dict) -> None: """倾听音乐""" genre =
params.get('genre', "classical") logger.info(f"Listening to {genre} music...")
print("[Robot: Playing beautiful music...]") self.nlp.play_music(genre) # 模拟机器人伴随音
乐的轻微动作 for i in range(5): self.joint_control.set_joint_angle(JointType.HEAD, 10.0 *
np.sin(i * 0.628)) time.sleep(1.0) def _read_newspaper(self, params: Dict) ->
None: """读报纸""" article = params.get('article', "Today's headlines")
logger.info(f"Reading newspaper: {article[:20]}...") print("[Robot: Reading the
newspaper aloud...]") self.nlp.read_newspaper(article) # 模拟翻页动作
self.joint_control.move_joints({ JointType.ELBOWS: 90.0, JointType.WRISTS: -15.0
}) self.finger_control.set_finger_position(1, 0.3) # 食指轻触 time.sleep(0.5)
self.joint_control.set_joint_angle(JointType.WRISTS, 15.0) # 翻页 time.sleep(0.5)# 主
系统类，整合所有模块 class ElderCareRobotSystem: def __init__(self): self.task_scheduler =
TaskScheduler() self.is_running = False def start_system(self) -> None: """启动机器人
系统""" self.is_running = True logger.info("Elder care robot system started")
print("=== 高级人形机器人老年护理系统已启动 ===") # 加载日常任务计划 self._load_daily_tasks() # 主循环
try: while self.is_running: time.sleep(0.1) except KeyboardInterrupt:
self.stop_system() def stop_system(self) -> None: """停止机器人系统""" self.is_running =
False logger.info("Elder care robot system stopped") print("=== 系统已停止 ===")
def _load_daily_tasks(self) -> None: """加载每日任务计划""" # 模拟早晨任务
self.task_scheduler.add_task("help_wash", {"type": "face"})
self.task_scheduler.add_task("help_dress", {"clothing_type": "shirt"})
self.task_scheduler.add_task("prepare_meal", {"meal_type": "breakfast"})
self.task_scheduler.add_task("feed_meal") # 上午任务
self.task_scheduler.add_task("have_conversation", {"topic": "yesterday"})
self.task_scheduler.add_task("do_laundry") # 中午任务
self.task_scheduler.add_task("prepare_meal", {"meal_type": "lunch"})
self.task_scheduler.add_task("feed_meal") # 下午任务 - 户外活动
self.task_scheduler.add_task("go_for_walk", {"distance": 3.0})
self.task_scheduler.add_task("sit_on_chair")
self.task_scheduler.add_task("listen_to_music", {"genre": "classical"})
self.task_scheduler.add_task("read_newspaper", {"article": "Today's news"}) # 晚
上任务 self.task_scheduler.add_task("prepare_meal", {"meal_type": "dinner"})
self.task_scheduler.add_task("feed_meal")
self.task_scheduler.add_task("give_medicine", {"pill_count": 2})
```

self.task_scheduler.add_task("help_go_to_bed") # 简化，实际应包含更多动作# 程序入口 if __name__ == "__main__": # 创建并启动机器人系统 robot_system = ElderCareRobotSystem() robot_system.start_system() 系统架构说明上述代码实现了一个完整的高级人形机器人老年护理系统，主要包含以下几个核心模块：1. 手指精细动作控制模块（FingerControl）：- 实现五指独立控制，精确到 0.0-1.0 的位置调节- 基于物体重量的智能抓握力调节- 完整的拾取物体动作序列，包括手指张开、移动、闭合等步骤 2. 关节运动控制模块（JointControl）：- 支持头部、肩膀、手肘、手腕、手指、躯干、髋关节等全身关节控制- 实现平滑的关节插值运动，确保动作自然流畅- 支持直线行走、角度运动等多种运动模式 3. 自然语言交互模块（NLPInteraction）：- 对话管理与历史记录- 简单的情感识别与响应调整- 音乐播放和报纸阅读功能 4. 任务调度模块（TaskScheduler）：- 任务队列管理与优先级处理- 完整的护理任务库，包括：- 生活照料：洗衣、拖地、做饭、喂饭、喂药等- 移动辅助：搀扶起身、行走、坐下等- 个人卫生：更衣、洗脸擦身等- 情感交互：对话交流、音乐欣赏、读报等- 机器人状态管理与任务状态转换 5. 主系统模块（ElderCareRobotSystem）：- 系统生命周期管理- 每日任务计划加载- 系统运行控制实际应用与技术扩展此代码为逻辑架构示例，实际应用中需要结合以下技术进行扩展：1. 硬件接口集成：- 使用具体机器人平台的 SDK（如优必选、波士顿动力等）- 集成传感器数据（视觉、力反馈、压力传感器等）- 实现基于传感器的实时动作调整 2. AI 技术增强：- 集成高级 NLP 模型（如 GPT 系列）实现更自然的对话- 加入计算机视觉进行环境感知与物体识别- 采用机器学习实现动作优化和老人状态识别 3. 安全与可靠性：- 加入紧急停止机制- 实现力反馈安全控制，防止对老人造成伤害- 增加故障检测与恢复功能 4. Firebase 集成：- 使用 Firebase 实时数据库存储老人健康数据- 通过 Firebase Cloud Messaging 实现远程监控与控制- 利用 Firebase Analytics 分析机器人使用情况要在实际机器人上运行此代码，需要根据具体的硬件平台调整关节控制接口和动作参数，并进行大量的动作调试以确保自然流畅和安全可靠。

要将数据库与机器人实际动作控制代码结合起来，一般可以按照以下步骤进行，以下以常见的机器人开发场景为例详细介绍：1. 确定通信接口与协议● 接口选择：根据机器人的硬件平台和控制需求，选择合适的通信接口，如串口通信（Serial）、以太网通信（Ethernet）、USB 通信等。● 协议制定：确定数据传输协议，例如 TCP/IP 协议用于网络通信，Modbus 协议用于工业设备通信。确保数据库与机器人控制代码之间能够准确、稳定地交换数据。2. 数据库操作封装● 数据库驱动安装：根据使用的数据库类型（如 MySQL、SQLite、PostgreSQL 等），安装相应的数据库驱动程序。例如，使用 Python 操作 SQLite 数据库可以使用内置的 sqlite3 模块，操作 MySQL 可以使用 mysql-connector-python 库。● 封装数据库操作函数：将数据库的基本操作（如查询、插入、更新、删除）封装成函数，方便在机器人控制代码中调用。以下是一个使用 Python 和 SQLite 数据库的简单封装示例：import sqlite3class DatabaseManager: def __init__(self, db_name): self.conn = sqlite3.connect(db_name) self.cursor = self.conn.cursor() def query(self, sql): self.cursor.execute(sql) return self.cursor.fetchall() def insert(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def update(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def delete(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def close(self): self.cursor.close() self.conn.close()3. 从数据库获取任务信息● 任务查询：在机器人控制代码中，通过调用数据库操作函数，查询需要执行的任务信息。例如，根据当前时间和任务表中的执行时间，筛选出需要执行的任务。# 初始化数据库管理器 db_manager = DatabaseManager('elderly_care_robot.db')# 查询当前时间需要执行的任务 current_time = '08:00'sql = f"SELECT actions.action_name, actions.description, tasks.scene FROM actions JOIN tasks ON actions.action_id = tasks.action_id WHERE tasks.execution_time = '{current_time}'"tasks = db_manager.query(sql)for task in tasks: action_name, description, scene = task print(f"当前任务：{action_name}，描述：{description}，场景：{scene}")4. 动作映射与执行● 动作映射：建立数据库中动作名称与机器人实际动作控制代码之间的映射关系。可以使用字典来实现这种映射，将动作名称作为键，对应的动作控制函数作为值。# 假设以下是机器人的动作控制函数 def feed_food(): print("正在执行喂饭动作")def feed_medicine(): print("正在执行喂药动作")# 动作映射字典 action_mapping = { "喂饭": feed_food, "喂药": feed_medicine}# 根据查询到的任务执行相应动作 for task in tasks: action_name = task[0] if action_name in action_mapping: action_mapping[action_name]() else: print(f"未找到对应的动作控制函数：{action_name}")5. 反馈与更新数据库● 执行反馈：在机器人执行完每个任务后，将执行结果反馈给数据库。可以在数据库中添加一个执行状态字段，记录任务的执行情况（如成功、失败、正在执行等）。● 更新数据库：根据执行反馈，更新数据库中的任务信息。例如，如果任务执行成功，将执行状态更新为“已完成”。# 假设任务执行成功，更新数据库中的执行状态 task_id = 1 # 假设任务 ID 为 1sql = "UPDATE tasks SET status = '已完成' WHERE task_id = ?"db_manager.update(sql, (task_id,))# 关闭数据库连接

db_manager.close()6. 异常处理与日志记录● 异常处理：在数据库操作和动作执行过程中，添加异常处理机制，确保系统的稳定性。例如，捕获数据库连接异常、SQL 执行异常等，并进行相应的处理。● 日志记录：记录数据库操作和机器人动作执行的详细信息，方便后续的调试和分析。可以使用 Python 的 logging 模块来实现日志记录。import logginglogging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')try: # 数据库操作和动作执行代码 passexcept Exception as e: logging.error(f"发生异常：{e}")通过以上步骤，可以将数据库与机器人实际动作控制代码有机地结合起来，实现机器人根据数据库中的任务信息执行相应动作，并将执行结果反馈给数据库的功能。

要将数据库与机器人实际动作控制代码结合起来，一般可以按照以下步骤进行，以下以常见的机器人开发场景为例详细介绍：1. 确定通信接口与协议● 接口选择：根据机器人的硬件平台和控制需求，选择合适的通信接口，如串口通信（Serial）、以太网通信（Ethernet）、USB 通信等。● 协议制定：确定数据传输协议，例如 TCP/IP 协议用于网络通信，Modbus 协议用于工业设备通信。确保数据库与机器人控制代码之间能够准确、稳定地交换数据。2. 数据库操作封装● 数据库驱动安装：根据使用的数据库类型（如 MySQL、SQLite、PostgreSQL 等），安装相应的数据库驱动程序。例如，使用 Python 操作 SQLite 数据库可以使用内置的 sqlite3 模块，操作 MySQL 可以使用 mysql-connector-python 库。● 封装数据库操作函数：将数据库的基本操作（如查询、插入、更新、删除）封装成函数，方便在机器人控制代码中调用。以下是一个使用 Python 和 SQLite 数据库的简单封装示例：import sqlite3class DatabaseManager: def __init__(self, db_name): self.conn = sqlite3.connect(db_name) self.cursor = self.conn.cursor() def query(self, sql): self.cursor.execute(sql) return self.cursor.fetchall() def insert(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def update(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def delete(self, sql, values): self.cursor.execute(sql, values) self.conn.commit() def close(self): self.cursor.close() self.conn.close()3. 从数据库获取任务信息● 任务查询：在机器人控制代码中，通过调用数据库操作函数，查询需要执行的任务信息。例如，根据当前时间和任务表中的执行时间，筛选出需要执行的任务。# 初始化数据库管理器 db_manager = DatabaseManager('elderly_care_robot.db')# 查询当前时间需要执行的任务 current_time = '08:00'sql = f"SELECT actions.action_name, actions.description, tasks.scene FROM actions JOIN tasks ON actions.action_id = tasks.action_id WHERE tasks.execution_time = '{current_time}'"tasks = db_manager.query(sql)for task in tasks: action_name, description, scene = task print(f"当前任务：{action_name}，描述：{description}，场景：{scene}")4. 动作映射与执行● 动作映射：建立数据库中动作名称与机器人实际动作控制代码之间的映射关系。可以使用字典来实现这种映射，将动作名称作为键，对应的动作控制函数作为值。# 假设以下是机器人的动作控制函数 def feed_food(): print("正在执行喂饭动作")def feed_medicine(): print("正在执行喂药动作")# 动作映射字典 action_mapping = { "喂饭": feed_food, "喂药": feed_medicine}# 根据查询到的任务执行相应动作 for task in tasks: action_name = task[0] if action_name in action_mapping: action_mapping[action_name]() else: print(f"未找到对应的动作控制函数：{action_name}")5. 反馈与更新数据库● 执行反馈：在机器人执行完每个任务后，将执行结果反馈给数据库。可以在数据库中添加一个执行状态字段，记录任务的执行情况（如成功、失败、正在执行等）。● 更新数据库：根据执行反馈，更新数据库中的任务信息。例如，如果任务执行成功，将执行状态更新为“已完成”。# 假设任务执行成功，更新数据库中的执行状态 task_id = 1 # 假设任务 ID 为 1sql = "UPDATE tasks SET status = '已完成' WHERE task_id = ?"db_manager.update(sql, (task_id,))# 关闭数据库连接 db_manager.close()6. 异常处理与日志记录● 异常处理：在数据库操作和动作执行过程中，添加异常处理机制，确保系统的稳定性。例如，捕获数据库连接异常、SQL 执行异常等，并进行相应的处理。● 日志记录：记录数据库操作和机器人动作执行的详细信息，方便后续的调试和分析。可以使用 Python 的 logging 模块来实现日志记录。import logginglogging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')try: # 数据库操作和动作执行代码 passexcept Exception as e: logging.error(f"发生异常：{e}")通过以上步骤，可以将数据库与机器人实际动作控制代码有机地结合起来，实现机器人根据数据库中的任务信息执行相应动作，并将执行结果反馈给数据库的功能。

要保证数据库和机器人动作控制代码结合时数据的实时性，可从数据库配置、数据交互、缓存策略、监控与错误处理等多个方面进行优化，以下是具体方法：数据库层面● 选择合适的数据库管理系统：不同的数据库在处理实时数据方面性能有所差异。例如，内存数据库（如 Redis）以内存作为数据存储介质，读写速度极快，能满足对实时性要求较高的场景；关系型数据库（如 MySQL、PostgreSQL）在数据一致性和事务处理上表现出色，可根据具体需求将 Redis 作为缓存层，MySQL 作为持久化存储。● 优化数据库配置：调整数据库的相关参数以提升性能。对于 MySQL 数据库，可以增大 innodb_buffer_pool_size 参数，让更多的数据和索引缓存在内存中，减少磁盘 I/O 操作；合理设置 max_connections 参数，确保数据库能够处理足够多的并发连接，避免因连接数不足导致数据处理延迟。● 数据库架构设计：采用主从复制或集群架构提高数据库的读写性能和可用性。主从复制模式下，主数据库负责写操作，从数据库负责读操作，将读请求分散到多个从数据库上，减轻主数据库的负担，提高数据读取的实时性。数据交互层面● 高效的数据查询与更新策略：在编写数据库查询和更新代码时，尽量减少不必要的查询和复杂的 SQL 语句。使用索引优化查询性能，确保查询能够快速定位到所需数据。例如，在经常用于查询条件的字段上创建索引。同时，采用批量操作的方式进行数据更新，减少与数据库的交互次数。● 实时数据推送机制：利用消息队列（如 Kafka、RabbitMQ）实现数据库与机器人动作控制代码之间的实时数据推送。当数据库中的数据发生变化时，触发相应的消息事件，将数据更新信息发送到消息队列中。机器人动作控制代码订阅消息队列，实时获取数据更新通知，并及时做出响应。● 优化网络通信：确保数据库与机器人控制设备之间的网络连接稳定、高效。采用高速网络，减少网络延迟。同时，对数据传输进行优化，例如使用压缩算法对数据进行压缩，减少数据传输量，提高传输速度。缓存策略层面● 使用本地缓存：在机器人动作控制代码中引入本地缓存机制，如使用 Python 的字典或第三方缓存库（如 cachetools ）。将经常访问的数据存储在本地缓存中，当需要获取数据时，先从本地缓存中查找，如果缓存中不存在再从数据库中获取，并将数据更新到缓存中。这样可以减少对数据库的频繁访问，提高数据获取的速度。● 缓存更新策略：制定合理的缓存更新策略，确保缓存数据与数据库数据的一致性。当数据库中的数据发生更新时，及时更新本地缓存中的相应数据。可以采用主动更新和被动更新相结合的方式，主动更新是指在数据更新后立即更新缓存；被动更新是指在缓存过期后重新从数据库中获取最新数据。监控与错误处理层面● 实时监控与报警：建立实时监控系统，对数据库的性能指标（如 CPU 使用率、内存使用率、查询响应时间等）和数据更新情况进行监控。当发现数据更新延迟或数据库性能下降时，及时发出报警通知，以便管理员及时处理。● 错误处理与重试机制：在数据交互过程中，可能会出现各种错误，如网络中断、数据库连接失败等。为了保证数据的实时性，需要在代码中添加错误处理和重试机制。当出现错误时，捕获异常并进行相应的处理，如重新连接数据库、重试数据操作等。同时，设置合理的重试次数和重试间隔，避免无限重试导致系统资源浪费。同步机制层面● 定时同步与实时同步结合：对于一些对实时性要求不是特别高的数据，可以采用定时同步的方式，定期从数据库中获取最新数据进行更新。而对于关键的、对实时性要求高的数据，则采用实时同步的方式，确保数据一旦更新就能立即被机器人动作控制代码获取到。● 版本控制与冲突检测：在数据更新过程中，引入版本控制机制，为每条数据记录一个版本号。当机器人动作控制代码获取数据时，同时获取数据的版本号。在更新数据时，比较版本号，如果版本号不一致，说明数据在获取后已经被其他操作更新过，此时需要进行冲突检测和处理，确保数据的一致性和实时性。

●●●（本文代码数据库仅供参考阅读，实战商战落地需要另行编程）。